SRCNN - Image Patch Upsampling

SPECTRUM

Rithvika Pervala, 11940830 Shubham Gupta, 11941140

Abstract

Upsampling of images has been an old technique used to increase sizes and make estimates for scenario that never existed. An estimation that seeks to tweak certain parameters to obtain results in an unknown direction. After using the SRCNN for creating HR images, we are venturing into unknown directions to see the changes that can be brought with upsampling in between. It is an altogether different technique used for checking resolution of an image and its enhancement

Adding data within data in the extrapolation process to give extensive results is documented. Exploring reverse pre-processing and taking a whole new level of image similarity indexes to check images have been taken into account

Previous Work

- Built a fully convolutional neural network (SRCNN) for image super-resolution. The network directly learns an end-to-end mapping between low to high resolution images, with little pre/postprocessing beyond the optimization.
- We established a relationship between our deep learning-based method and the traditional sparse-coding-based methods. This relationship provides a guidance for the design of the network structure.
- We demonstrated that deep learning is useful in the classical computer vision problem of super resolution, and can achieve good quality and speed.
- The concepts of the **Mean Square Error** loss, **Stochastic Gradient Descent** model optimizer and **Rectified Linear Unit** activation function were used.

Introduction

Upsampling is defined in the context of Digital Signal Processing. It was then later on extended to digital photography systems. It is essentially defined as zero stuffing of signals onto the main sample of a signal to give an output which is obtained if the sampling rate of the same signal is enhanced. If the size of the signal goes up, then we have blank spaces or zero spaces that get placed. Upsampling stuff this empty spaces to eliminate noises present in between to give a better sample signal after noise removal.

Upsampling in simple forms can be defined as expansion of a sample to accommodate a scenario on a scale which is bigger than it’s current scale. It is an estimation model that approximates the current image size to an image whose size has been scaled to a larger factor than the previous factor. In digital photography, upsampling is generally used in post processing phase to increase the resolution of images. The main purpose of such techniques is to make images smoother during printed and display to the user, and reduce pixilation and other visual appeal factors that decrease the graphical display to the user.

Definition of the Problem

Enhance the resolution of images by upsampling certain regions within images to compound the output of the SRCNN model

---

Objective

- Upsample certain regions in High-Resolution Images and see the impact of upsampling in the quest of clearer High-Resolution Images.
- Post-process images after the output of the SRCNN model by extracting certain regions and apply upsampling techniques on them.
- Use multiple image similarity metrics and prepare a comprehensive analysis on how each similarity metric uses contrasts the upsampled and input images

---

Technology Used

- Google Colab
- NVIDIA GPU CUDA 4.0
- Google Drive
- Github
- Image Reader

Libraries Used

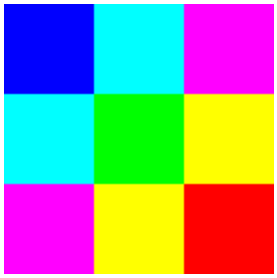
- Standard Python Libraries for I/O and
- System Operations
- python=3.6.8
- h5py==2.10.0
- ipykernel==5.1.3
- Markdown==3.1.1
- matplotlib==3.1.2
- ipython==7.9.0
- ipython-genutils==0.2.0
- jupyter-client==5.3.4
- jupyter-core==4.6.1
- tensorboard==2.0.1
- torch==1.3.1
- torchvision==0.4.2

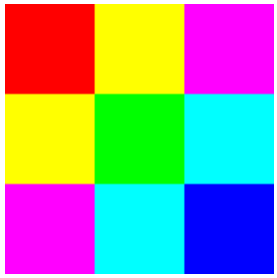
- numpy==1.17.4
- scikit-image==0.16.2
- scipy==1.3.3
- tqdm==4.40.0
- image-similarity-measures==0.3.5
- opencv-python==4.5.4.60

Problems Faced

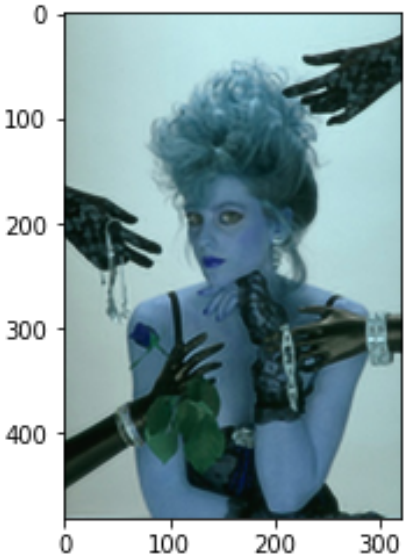
OpenCV Default Colour Space (Red-Blue Inversion)

- OpenCV assumes any image provided to be **BGR or BGRA** by default instead of classical RGB This means that blue and red planes get flipped.

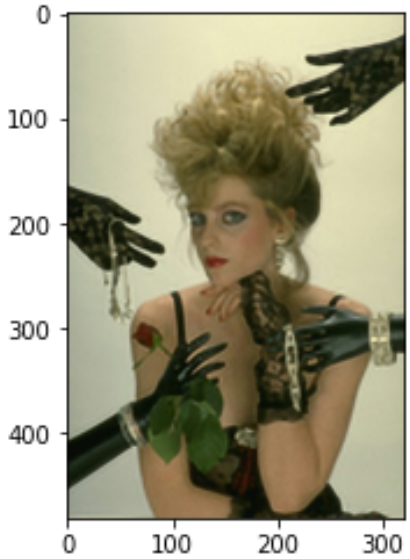




- So while using the command `cv2.imwrite(<image_name>, <image>)` to save our SRCNN output images, we ended up saving the below image and didn't realize until the Upsampling Phase. These images are now stored in `Red-Blue` directory.
- To compensate this we used `cv2.imwrite(<image_name>, cv2.cvtColor( <image>, cv2.COLOR_RGB2BGR))` .



Inverted Output Image



Correct Output Image

Evaluation Metrics Time Complexity

- Image Similarity Metrics given below are not used as they needed indefinite amount of time to compute despite importing them from various packages or even making custom functions
  - UIQ **(Universal Image Quality Index)**
  - ISSM **(Information theoretic-based Statistic Similarity Measure)**
  - FSIM **(Feature Similarity Indexing Method)**
- These metrics took very long on running on evaluation. A single iteration of running these metrics on a couple of images caused the RAM to exceed it's limit (perhaps beyond swap space as well) and a session crash.
- It will take too long to run them for over 4000+ comparisons which will definitely lead to memory and time issues.

Upsampling Technique Used

Bicubic Interpolation Upsampling

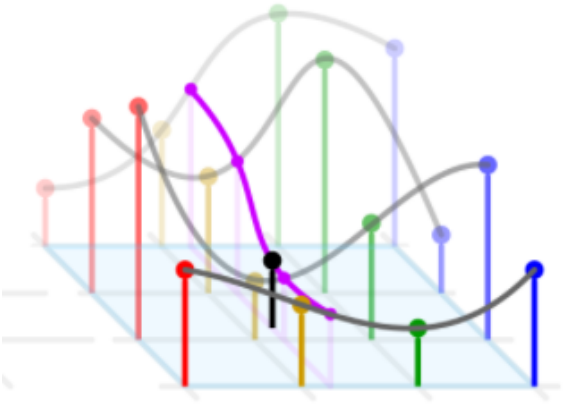
- Bicubic upsampling is a technique to sharpen and enlarge digital images. It is a technique to enlarge the size of the images, so it could lead to loss in details, so each pixel has to be approximated with its surrounding pixels in order to get the closest value. Pixel addition must be accurate enough to recreate all the details of the image and sharpness needs to be retained.

$$p(x,y) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}x^i y^j$$

a_{ij} - Vector parameters

$p(x,y)$ - Point

- For a particular image which is represented as a grid, we take the values of a point at $p(x,y)$ and interpolating to approximate to the nearest point. This is done iteratively for all the points in the graph. This function directly implies that on upsampling an image by a large size, we need to preserve the details for more iterations.
- The higher the resolution of the input images leads to preservation of more details during he upsampling leading to a better resolution in contrast with the initial image. It also smoothens the image in a manner making it easier to process it later.
- It is very frequently used in upsampling images in post processing in videos and images.



Bicubic

Implementation Procedure

Reverse-Preprocessing/Deprocessing

- To make the images ready for model training, we apply the **pre-processing** steps on them.
- However, we now need to upsample certain regions of them past the model training. So to do this, the current state of the model needs to be saved in it's current torch state to be applied on the SRCNN images.

Un-Standardization

- We obtained the following values of standard meaning and deviation in the previous part which were essentially subtracted followed by division respectively.

```
0.43398995682829317
0.23199219136380714
```

Standard Mean and Deviation

- The reverse is done i.e. we multiple the standard deviation followed by addition of the standard mean. As the dataset remains the same, the values remain the same

Un-Normalization

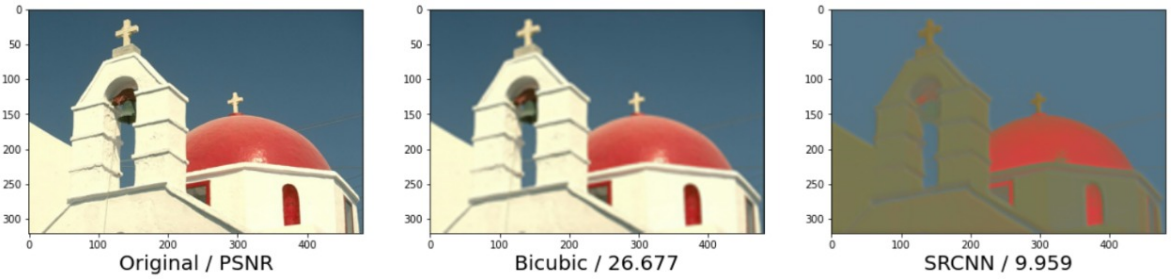
- We divided the image colour space into the regular size of 256 for model processing in the previous part.
- The reverse is done here where integrate the entire size of the colour space to form one ground truth image.

Colour space Integration

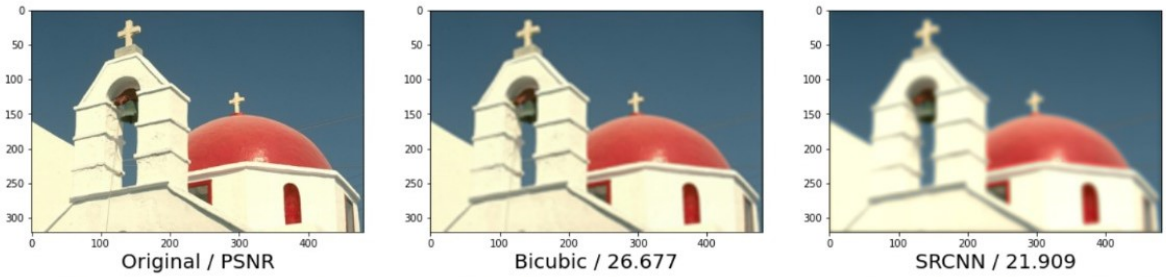
- Previously, we had converted the colour space from $\mathbf{RGB} \rightarrow \mathbf{YC_bC_r}$ and split each colour space into a particular dimension of the image. We then pre-trained the model on the Y -Luminance Channel only.
- As the colour space was split during the pre-training, we link all aspects of the $\mathbf{YC_bC_r}$ $(\mathbf{Y}, \mathbf{C_b}, \mathbf{C_r})$ colour space for the image to be ready.
- This means that each color space aspect in one dimension leads to 3 different dimensions. We bring all the 3 dimensions into a single one and generate the output $\mathbf{YC_bC_r}$ colour space.
- Since, there is an initial conversion from $\mathbf{RGB} \rightarrow \mathbf{YC_bC_r}$ colour space, the reverse is done by converting the $\mathbf{YC_bC_r} \rightarrow \mathbf{RGB}$ colour space again.

Image Saving

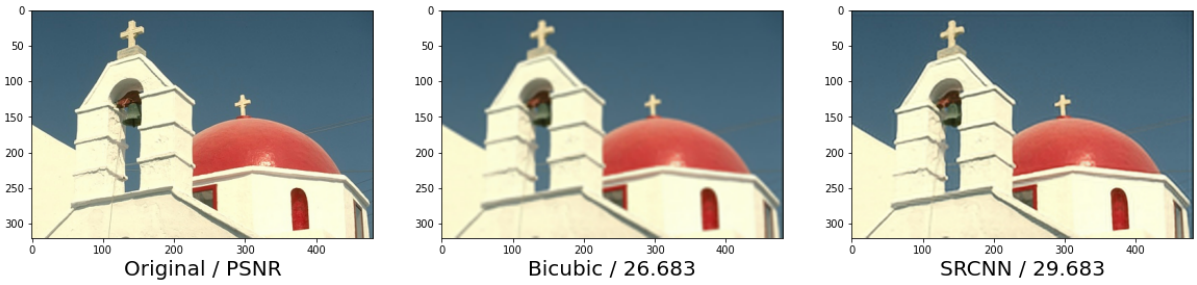
- We execute the processing on each epoch of the model and save the images in a directory to apply upsampling techniques on them.
- We can also plot these integrated images and see the progress of the SRCNN model over different epochs.



Epoch 3



Epoch 15



Epoch 9000

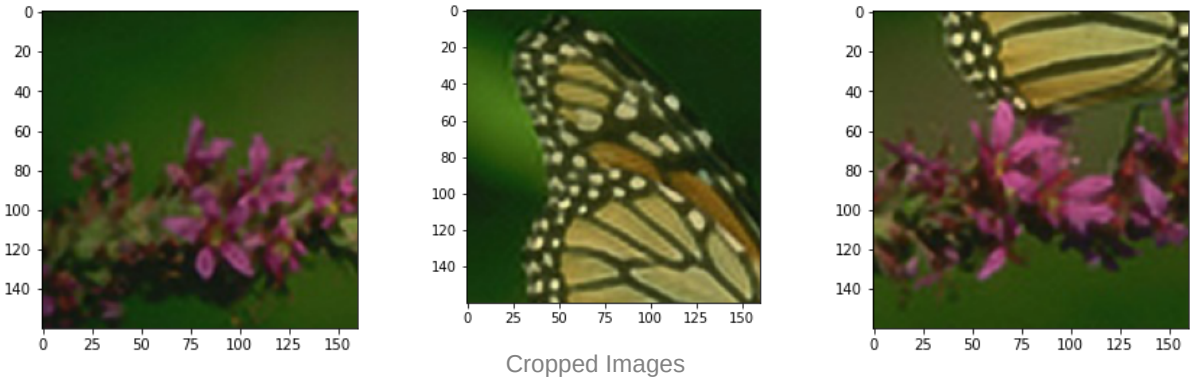
- We directly process the images the images for upsampling on the SRCNN resolution till that epoch/point. We perform the iterative blurring in each epoch and we observe the effect of upsampling as the images are deblurred after each point. On moving onto the next phase, we can see the change in the images obtained by the SRCNN as we proceed towards the upsampling phase.
- These images are then stored in the folders `Test_SRCNN_Output` for test set and `Eval_SRCNN_Output` for evaluation set.

Upsampling phase

- In images, upsampling deals with the increasing the spatial resolution of while keeping the dimensions same. The colour coding and matrices are retained, with resolution enhanced.
- In the process of upsampling, dimensions are increased that leads to a lot of pixels staying empty. Upsampling fills these **empty pixels**, a term known as **Zero-stuffing** in digital photography, with approximate and estimated data based on the input image.
- In our case we take the output images generated from the SRCNN Model after 9000 epochs and iteratively crop the images using a custom function `get_image_crops()` from the **Postprocessing** Folder.

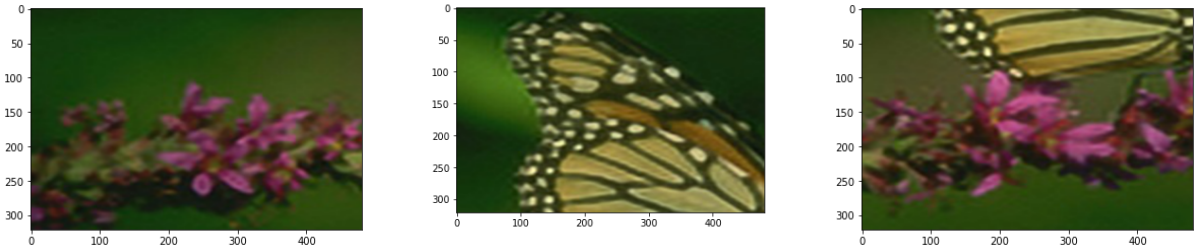
SRCNN Output

- `get_image_crops()` gets the crops by iteratively striding over the submatrices of given height and width (default 160 pixels each) and then returning these crops.
- Since the Output images are either 480×320 or 320×480, we will get a total of 6 crops per image with the default crop height and width (160)



Cropped Images

- We then use these Cropped Patches and upsample them using **bicubic interpolation.** We do that by using another custom function `upsample_images()` from the **Postprocessing** Folder.
- Other upsampling techniques like **Bilinear Interpolation** can also be implemented, but we chose to go with Bicubic



Upsampled Crops

Evaluation Phase

- After the upsampling of all the crop patches of the output images are done, we compare these patches with the output image using 6 **image similarity metrics**
- All of these metrics are imported from the python package `image-similarity-measures`
- Once the upsampling is complete, we use these 6 metrics to compare between the upsampled segment of the image and the original image.

PSNR (Peak Signal Noise Ratio)

- PSNR is a widely used metric for quantitatively evaluating high **restoration quality.**
- It measures the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation.
- Partially related to **perceptual quality** as well
- PSNR is usually expressed as a logarithmic quantity using the **decibel scale (dB)**.
- MSE is known to favour High PSNR value. It's value based on MSE can be derived using the following equations

$$\mathrm{MSE} = \frac{\sum_{M,N}[I_1(m,n) - I_2(m,n)]^2}{M * N}$$

$$\mathrm{PSNR} = 10\log_{10}(\frac{R^2}{\mathrm{MSE}})$$

R - Maximum possible pixel value

I_1 - Original Image

I_2 - Predicted Image

M - Number of Rows

N - Number of Columns

SSIM (Structural Similarity Index Measure)

- SSIM is used to quantify **Perceptual Quality -** How humans perceive it.
- Quantifies image quality degradation via, visible image structures, caused by processing - lossy data compression, lossy data transmission.
- It ranges between $[-1, 1]$

- The measure between two windows x and y:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

μ_x - Average of x

μ_y - Average of y

σ_x - Variance in x

σ_y - Variance in y

σ_{xy} - Covariance in x, y

$c_1 = (0.01 \times L)^2$

$c_2 = (0.03 \times L)^2$

L - Dynamic Range of Pixel Values

- Although this metric better evaluated over MAE **(Mean Absolute Error),** MSE also gives satisfactory performance over SSIM**.**

RMSE (Root Mean Square Error)

- RMSE measures the amount of change per pixel due to the processing.
- RMSE range from $[0, 1]$ with 0 meaning being the images being compared are identical.
- The RMSE value can be calculated using the below equation-

$$\mathrm{RMSE} = \sqrt{\frac{1}{M * N} \sum_{i=0,j=0}^{M-1,N-1} [I_1(i,j) - I_2(i,j)]^2}$$

$I_1(i,j)$ - Original Image

$I_2(i,j)$ - Predicted Image

M - Number of Rows

N - Number of Columns

SRE (Signal to Reconstruction Error Ratio)

- SRE measures the error relative to the power of the signal.
- Using SRE is better suited to make errors comparable between images of varying brightness. Whereas the popular PSNR would not achieve the same effect, since the peak intensity is
  constant.
- SRE similar to PSNR uses decibel scale and is computed using the following equation

$$\mathrm{SRE} = 10 \log_{10} \frac{\mu_x^2}{|\hat{x} - x|^2 / n}$$

μ_x - Average of x

SAM (Spectral Angle Mapper)

- SAM is physical based spectral classification. The algorithm determines the spectral similarity between two spectra by calculating the angle between the spectra and treating them as vectors in a space with dimensionality equal to the number of bands.
- SAM value is determined by applying the following equation

$$\alpha = \cos^{-1} \frac{\sum_{i=1}^{nb} t_i r_i}{\sqrt{\sum_{i=1}^{nb} t_i^2} \sqrt{\sum_{i=1}^{nb} r_i^2}}$$

t, r - Image Pixel Spectrum

n - Number of available spectral bands

b - Bandwidth

COS (Cosine Similarity)

- Quantifies the image similarity in terms of pose angle.
- It is essential the Euclidean Dot Product of the two images.
- It ranges between $[-1, 1]$
- It can be calculated using the below equation for two images A and B

$$\mathrm{COS} = S_C(A, B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Other evaluation metrics

- These metrics took very long on running on evaluation. A single iteration of running these metrics on a couple of images caused the RAM to exceed and a session crash.
  - UIQ **(Universal Image Quality Index)**
  - ISSM **(Information theoretic-based Statistic Similarity Measure)**
  - FSIM **(Feature Similarity Indexing Method)**
- It will take too long to run them for over 4000+ comparisons which will definitely lead to memory and time issues.

Results - 1 HR Image Output

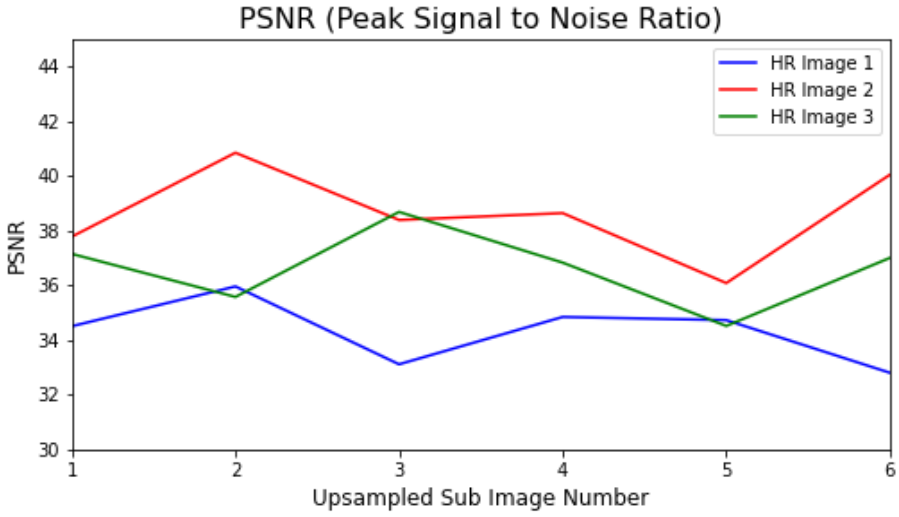
Sub Image	PSNR	SSIM	RMSE	SRE	SAM	COS
1	40.868	0.88	0.019	45.639	0.009	6.814×10^{-9}
2	39.148	0.856	0.011	45.031	85.915	6.311×10^{-8}
3	39.736	0.847	0.0103	44.944	86.315	6.302×10^{-8}
4	40.855	0.885	0.0091	45.569	87.372	4.105×10^{-8}
5	39.236	0.849	0.0109	44.89	86.54	1.094×10^{-7}
6	38.791	0.816	0.0115	44.527	85.559	5.64×10^{-8}

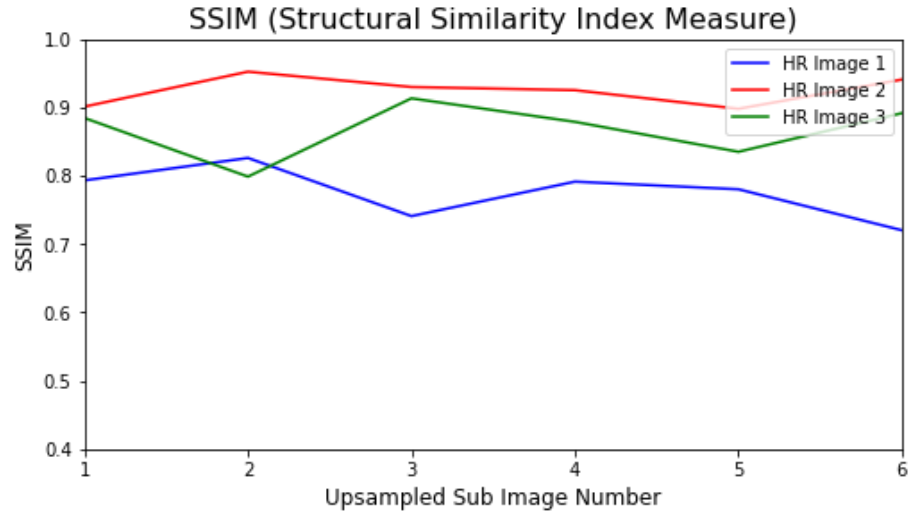
```
Image 1
psnr - [40.867898499278596, 39.14791825941043, 39.73576072385745, 40.8547616355701, 39.23646453550296, 38.790521656779816]
ssim - [0.8800768344684115, 0.8556641623509454, 0.8465581806506858, 0.8846859889907824, 0.8492751219316972, 0.8160473854674017]
rmse - [0.009049092, 0.011030726, 0.010308892, 0.00906279, 0.010918848, 0.011494073]
sre - [45.63894224887463, 45.030926867668654, 44.94374052225844, 45.56906511259253, 44.890290137718964, 44.527010958199455]
sam - [85.99681963983153, 85.9150428311379, 86.315288756574, 87.37172245163302, 86.54040077205036, 85.55940725955492]
cos - [7.594140081788097e-08, 4.802693143244291 5e-08, 4.126704921165863e-08, 4.540844113303519e-08, 1.1463932269894362e-07, 3.512061646618177e-08]
```

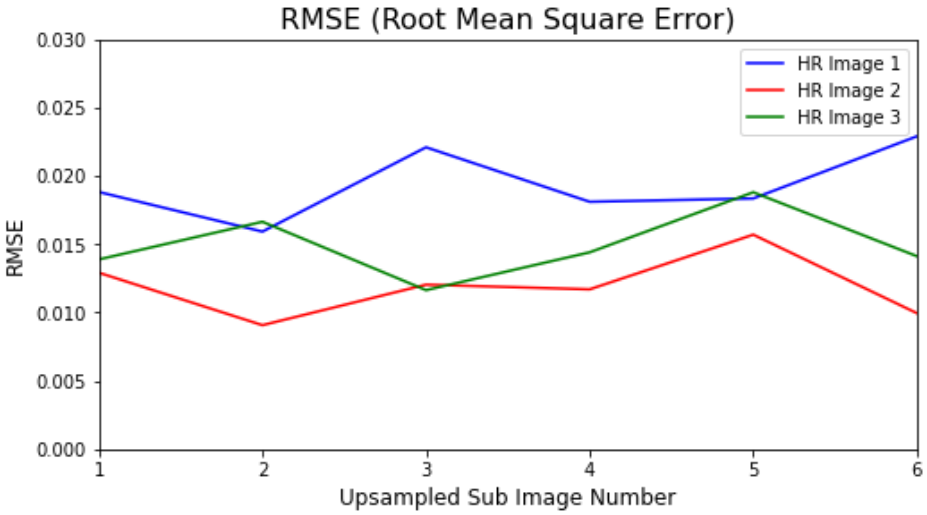
Evaluation Metric Values for one image

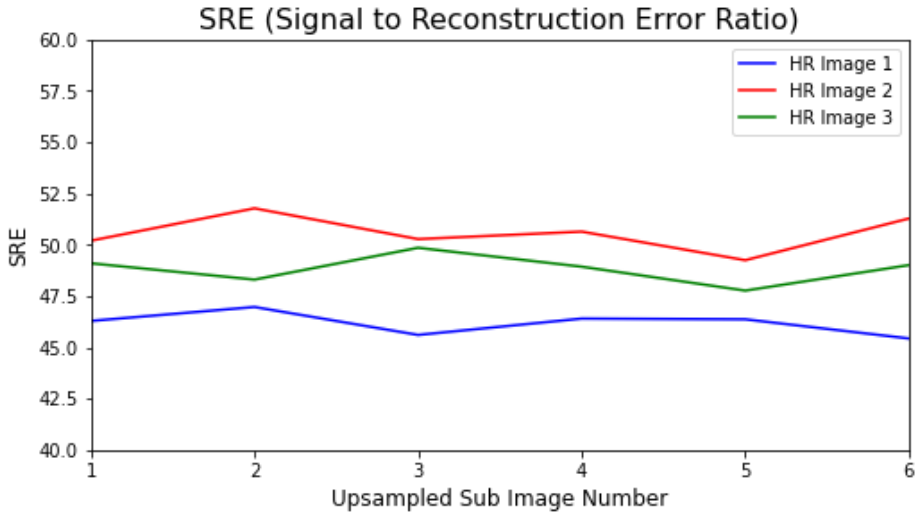
Results and Analysis

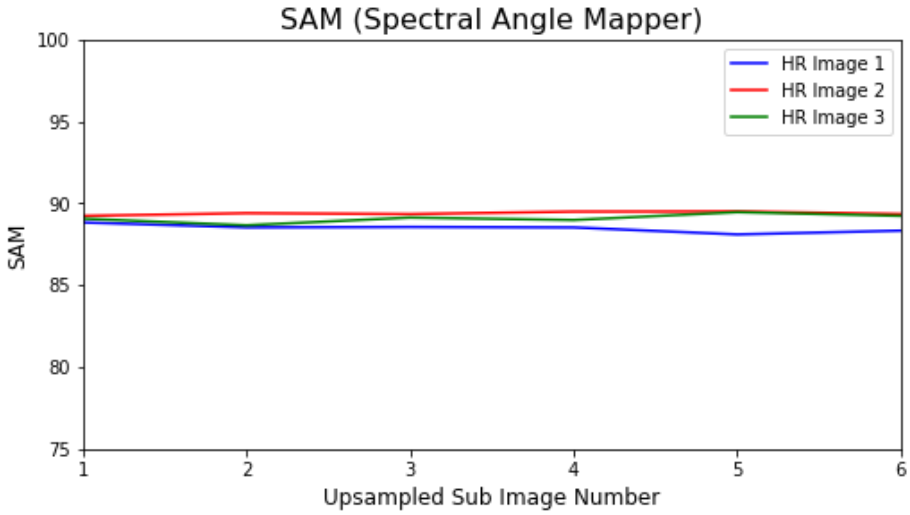
- Using a custom function `upsampling_plot()` from `Postprocessing/plots.py` we plot the above found results for 3 High Resolution Images and their Upsampled sub patches across 6 different metrics the following way.

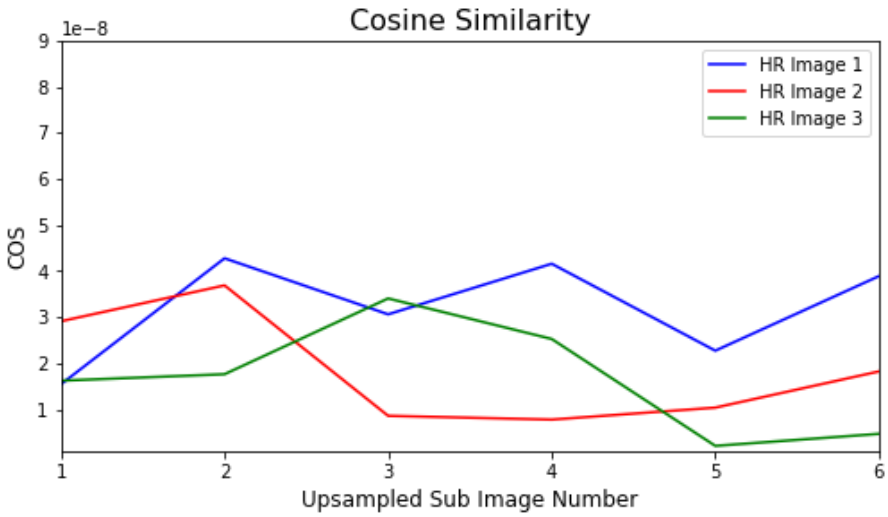



SSIM (Structural Similarity Index Measure)
HR Image 1
HR Image 2
HR Image 3
SSIM
1.0
0.9
0.8
0.7
0.6
0.5
0.4
1
2
3
4
5
6
Upsampled Sub Image Number


RMSE (Root Mean Square Error)
HR Image 1
HR Image 2
HR Image 3
RMSE
0.030
0.025
0.020
0.015
0.010
0.005
0.000
1
2
3
4
5
6
Upsampled Sub Image Number


SRE (Signal to Reconstruction Error Ratio)
HR Image 1
HR Image 2
HR Image 3
SRE
60.0
57.5
55.0
52.5
50.0
47.5
45.0
42.5
40.0
1
2
3
4
5
6
Upsampled Sub Image Number





Conclusion

We showed 3 major steps

- Reverse Processing/Deprocessing
  - Saw the impact of Deprocessing iteratively while extracting the deblurred images after every epoch of the training model. We saw the image comparison and brightness/resolution after every epoch and the difference in PSNR during the deblurring process.
- Cropping followed by Upsampling
  - Saw the upsampling in each iterative crop of the SRCNN output image and noticed the difference in the resolution pre and post upsampling.
- Comprehensive Analysis

- Built a comprehensive image similarity comparison using a host of metrics and their library implementations to see a vivid comparison across the metrics.
- Plotted graphs to give a better graphical understanding of the upsampling behaviour in our model.

References

Papers

- https://arxiv.org/pdf/1501.00092.pdf
- https://www.mdpi.com/2076-3417/11/3/1092/pdf
- https://paperswithcode.com/paper/image-super-resolution-using-deep

Tech Links

- https://up42.com/blog/tech/image-similarity-measures
- https://clouard.users.greyc.fr/Pantheon/experiments/rescaling/index-en.html
- https://kiwidamien.github.io/how-to-do-cross-validation-when-upsampling-data.html
- https://www.geeksforgeeks.org/spatial-resolution-down-sampling-and-up-sampling-in-image-processing/
- https://medium.com/hd-pro/bicubic-interpolation-techniques-for-digital-imaging-7c6d86dc35dc
- https://en.wikipedia.org/wiki/Bicubic_interpolation
- https://www.cs.toronto.edu/~guerzhoy/320/lec/upsampling.pdf
- https://stackoverflow.com/questions/67957965/how-to-perform-bicubic-upsampling-of-image-using-pytorch

Repositories

- https://github.com/up42/image-similarity-measures
- https://github.com/highgroundmaster/Image-Super-Resolution-Using-Deep-Convolutional-Networks
- https://github.com/sdv4/SRCNN
- https://github.com/princepansari/Video-SuperResolution_And_GAN_FakeImageDetection
- https://github.com/sdv4/FDPL
- https://keras.io/examples/vision/super_resolution_sub_pixel/
